**Introduction**

In this module, you connect a web application to a database. You manage changes to the database schema in Azure Pipelines.

In the Create a multistage pipeline by using Azure Pipelines  module, you helped the Tailspin Toys web team design and build a multistage release pipeline. The team uses the pipeline to move changes through a series of stages. Changes move through the *Dev* stage, the *Test* stage, and finally the *Staging* stage, which resembles a production environment.

The stages that you and the team defined provide the overall shape of the pipeline. But you can add more to each stage. For example, in the *Dev* stage you can verify that multiple components, such as the web application and a database, work as expected in a real environment.

Currently the web application reads test data from local files. In this module, you connect the website to Azure SQL Database and load your database with sample data.

When an application uses a database, the database administrator (DBA) and the app developers work together to design and maintain the *database schema*. Think of the schema as a blueprint that dictates how the data will be organized. Azure DevOps can help developers and DBAs coordinate their plans and deploy database schema changes quickly and safely.

**Learning objectives**

In this module, you:

* Connect a relational database to your website. For this example, you use Azure SQL Database and Azure App Service.
* Coordinate changes to the database schema by using a release approval.
* Use PowerShell to share variables among pipeline stages.

**Prerequisites**

The modules in this learning path form a progression. To follow the progression from the beginning, start with these learning paths:

* Evolve your DevOps practices
* Build applications with Azure DevOps

We also recommend that you start at the beginning of the Deploy applications with Azure DevOps  learning path.

If you want to go through just this module, set up a development environment on your Windows, macOS, or Linux system. You need these assets:

* An Azure DevOps organization .
* An Azure subscription .
* A GitHub  account.
* Visual Studio Code .
* .NET Core 3.1 SDK .
* Git .

You can get started with Azure and Azure DevOps for free. You don't need an Azure subscription to work with Azure DevOps. But here you'll use Azure DevOps to deploy to resources that exist in your Azure subscription.

In this environment, you'll complete the exercises in this and future modules. You can also use it to apply your new skills to your own projects.

**Note**

Keep in mind that you can use Azure DevOps to build and deploy almost any kind of application written in any language. In this module, you'll be working with a .NET Core application written in C#.

You don't need to be an expert in .NET or C# to complete this module. You can apply the patterns you learn here to your own projects that use your favorite programming languages and frameworks.

**Meet the team**

You met the *Space Game* web team at Tailspin Toys in previous modules. As a refresher, here's who you'll work with in this module.



Andy is the development lead.



Amita is in QA.



Tim is in operations.



Mara just joined as a developer and reports to Andy.

Mara has prior experience with DevOps. She's helping the team adopt a more automated process that uses Azure DevOps.

# Design a database schema-change workflow in Azure Pipelines

* 7 minutes

The team has decided it's time to connect their website to a live database. To coordinate the work, they need to work with the Tailspin DBA. The DBA is responsible for maintaining the integrity of the database. That person will work with the team to understand what they need and to design a schema. If the team later wants to change the schema, the DBA needs to approve the changes before they can take effect.

Now that the team understands the benefits of automation, they want to make the approval process a part of their release pipeline.

In this section, you:

* Choose a database technology for your application.
* Learn how to configure Azure App Service to use a database connection string.
* Understand the role of a SQL Server Data Tools project in Azure Pipelines.
* Learn how to approve database schema changes in Azure Pipelines.

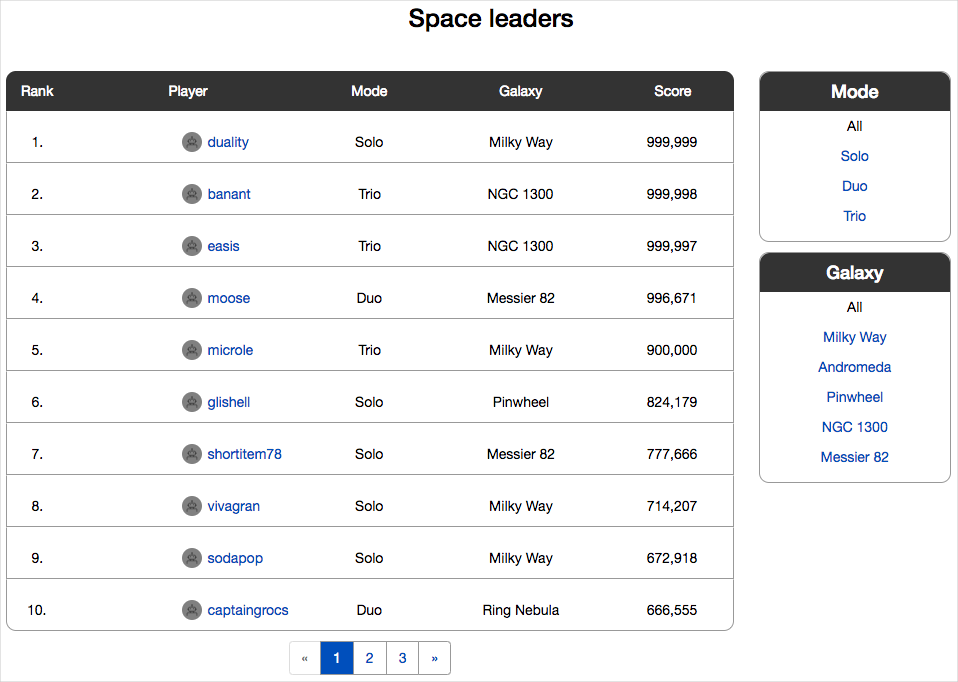
## The meeting

The team deploys their web app through a series of stages. In the Dev stage, all of the development pieces come together. In the Test stage, QA performs tests. In Staging, management can approve the app before it moves to production.

Here the team discusses how to add a database as the data storage solution for the web app. They plan to integrate the database and the web app in the Dev stage.

**Andy:** Good morning. We've all seen how Azure Pipelines has accelerated our release process and improved the quality of what we do. Because we've made so much progress, we're ready to connect the website to a real database. I think this integration can happen as early as the Dev stage. By connecting the website to the database early in the pipeline, we can track changes and measure performance all the way through the pipeline. Ultimately, we want the leaderboard to pull its data from the database. Here's what the leaderboard looks like now.

Andy shows the Space Game site that's running on Azure App Service.



**Andy:** After the database is in place, we can modify the website to read from the database instead of from local files. This setup is a significant milestone for us, so we need to discuss a few things. First, let's choose a database technology. We'll need to consider numerous things, so bear with me.

## Choose a database technology

When considering a data storage solution, you can choose from several options. The following factors can help you choose.

### The type of data you're storing

Data can be structured, semistructured, or unstructured:

* Structured data is relational. It fits into the columns and rows of a table. For example, think of a point-of-sale system. You might have a table for product data whose columns define how you describe your products. For example, columns can be named **Product Name**, **SKU**, **Color**, and so on.
* Unstructured data includes documents such as text files, photos, and videos. This data format is for quick access to an entire file and not for processing records.
* Semistructured data is nonrelational. It fits into hierarchies by using tags. This format is good for data whose description might change. Your product data is semistructured if you can't use the same columns to describe all of your products the way you can with structured data. Usually, this data is stored in a JSON or XML file format.

### Your operational needs

As your team plans your operational needs, ask yourselves:

* Will we be doing basic lookups? Or do we need to join queries from multiple datasets?

Basic lookups work with nonrelational data. Join queries require relational data.

* How often will the data change?

Reporting data might change only at end of the month or quarterly. This kind of data could be a candidate for a data warehouse. If the data changes throughout the day, you need something that can handle fast transactional throughput.

* Do we always need the latest data?

Perhaps the data changes often, but you need it only at the end of each month. Or perhaps you trade stock and need to fetch current data as quickly as possible.

* Do we need to run complex analytical queries?

Complex analytical queries are queries that can be done on several dimensions. If you need to run this sort of query, a data warehouse is a good choice.

* How quickly do these operations need to finish?

Do customers consume the data directly? If they do, the data operations need to be fast. Perhaps the data is used to run nightly reports. In this case, operations can run more slowly.

### Transactions

You need transactions when your data has relationships and those relationships keep the data in sync or help you to query the data. For example, let's say that you upgrade your gym membership to receive more privileges. Your account is updated to charge you more per month. But the corresponding update to your membership profile fails. This failure might cause you lose access to the privileges that you pay for. A transaction would ensure that both pieces of data are updated at the same time. If either update fails, the entire transaction is rolled back.

Your data might need one of these two types of transactions:

* Online Transaction Processing (OLTP) is used for relational data. You would use OLTP to solve the problem in the preceding example.
* Online Analytical Processing (OLAP) is used for more complex queries. For example, use OLAP when your data is in a structure called a cube or data warehouse. OLAP is for data analysis systems that query on many dimensions of data. For example, say you want to know how many blue widgets were sold in January of last year in the northeast region. You want to compare that number with how many green widgets were sold from May through June of this year in the southwest. You also want to know the top salespeople for widgets in each region.

**Note**

At the end of this module, you'll find resources to help you choose a data storage option.

## The team's choice

Let's listen in on how the Space Game web team decides which data storage option to use.

**Tim:** As we make our choice, we also need to consider geo-replication and backups. We had a good experience deploying the web application to Azure App Service. What are our options for running a database on Azure?

**Andy:** I know that Azure SQL Database is a popular way to work with structured data. Azure Cosmos DB works with both unstructured and semistructured data. We could also use Azure Blob storage to manage unstructured data, like images.

**Mara:** The Space Game leaderboard data is relational. Profiles, scores, and achievements all relate to one another. The web application queries this data each time a user loads the home page, so quick retrieval of data is important. Those two qualities should help us narrow down our options.

**Amita:** The data changes constantly as people play the game. Players want to see their positions on the leaderboard as quickly as possible, before they get outranked by other players.

**Tim:** I've had a few conversations about Cosmos DB with our DBAs. They want to try it out. But I don't think we have time ramp up for this project. We know that our website uses relational data. We know that the overall shape of the data won't change much. And we know we need fast retrieval times. I think Azure SQL Database is the right choice.

**Mara:** That sounds good. I've used SQL Server Data Tools to create a database project. I'll add a database project to our Tailspin.SpaceGame.Web solution. The project will have the SQL scripts that we need to create the tables. We can use it to make sure that changes to the database schema in the project are checked against the current database schema.

**Andy:** Great. Adding the database project will be your action item. But now I have another question. If we create the database and connect to it from our website code, how do we keep the database connection string secure? I don't want to keep it in our appsettings.json file because that file will become part of our GitHub repository.

## Configure the web app to use a database connection string

The Space Game web application currently reads leaderboard data from JSON files. The team plans to convert the application to read leaderboard data from a database.

Recall that the Space Game web application uses ASP.NET Core and is written in C#. The source code defines the IDocumentDBRepository interface to fetch leaderboard data. The LocalDocumentDBRepository class implements IDocumentDBRepository to read from local JSON files.

Mara wants to update the code to read from a database instead of from files. So Mara creates another implementation of IDocumentDBRepository that connects to Azure SQL Database. This code gets the connection string from the website configuration.

C#Copy

public class RemoteDBRepository : IDocumentDBRepository

{

private readonly IConfiguration configuration;

private readonly string connectionString;

public RemoteDBRepository(IConfiguration config)

{

configuration = config;

connectionString = configuration.GetConnectionString("DefaultConnection");

}

The connection string uses SQL Authentication, which includes the username and password. If Mara stores this information in plain text in appsettings.json, anyone who can access this file can read the username and password. Instead, Mara uses the Secret Manager tool in Visual Studio. The tool stores the string in a file that isn't maintained in source control. Let's listen in as she explains the decision to the team.

**Mara:** When we're developing the app locally, we can use a file that's named secrets.json. This file isn't pushed to GitHub. We set up the file in Visual Studio by using the Secret Manager tool. But when the deployed web app needs the username and password, they'll be in the Azure App Service appsettings.json file. App Service has strict limited permissions for who can see the file. We can provide the information to App Service through Azure Pipelines. So we won't need to add it to our local appsettings.json file.

**Andy:** Good idea, Mara. We're making progress.

**Tim:** Don't start celebrating yet. We need a plan for when the database schema changes. Mara mentioned that the schema might change. How do we make sure that the DBA is happy with the changes and that the changes happen at the right time?

In this short video, Abel Wang, Cloud Advocate at Microsoft, explains the concept of database changes in Azure Pipelines.

**Ask Abel**

## The role of the SQL Server Data Tools database project in Azure Pipelines

SQL Server Data Tools, which runs on Windows, provides a project type that you can use to define a database schema from Visual Studio. This kind of project produces a dacpac file. When you unpack the dacpac, you see the SQL scripts that create the database schema. For example, you might see a CREATE TABLE script for each table that the database project defines. SQL Database can unpack the dacpac and apply the schema changes.

The database project open in Visual Studio on Windows.

Let's go back to the team discussion and see how they plan to handle any changes to their database schema.

**Mara:** The SQL Server Data Tools project that I mentioned can help. It creates a file, called a dacpac, that contains the current schema. We use this file to deploy the schema. Let's see if Azure Pipelines can create a file that shows the differences between the current database schema and the changes that the dacpac proposes. Then the DBA can look at the changes and approve them. We just got release approvals working. We'll try to take advantage of the release approvals to automate as much as possible.

Andy types on his laptop.

**Andy:** I found an Azure Pipelines task that we can use. The SqlAzureDacpacDeployment@1  task generates a file that identifies the schema differences between the current database schema and the dacpac.

## Approve database schema changes in Azure Pipelines

After you create the database change file, you can use PowerShell to write its contents to the pipeline so that the DBA can see the changes. Here's an example:

**yml**

- task: PowerShell@2

displayName: Show auto-generated SQL Script - check for schema changes

inputs:

targetType: 'inline'

script: |

Write-Host "Auto Generated SQL Update Script:"

Get-Content d:\a\1\s\GeneratedOutputFiles\$(databasename)\_Script.sql | foreach {Write-Output $\_}

You can use a manual approval to pause the pipeline at the stage where the changes happen, just as you did in the Create a multistage pipeline by using Azure Pipelines  module. You create an Azure Pipelines environment that specifies the DBA as the approver. If the DBA approves the changes, the pipeline continues and the changes are applied to the database. If the DBA rejects the changes, the pipeline is paused. Then you can discuss the proposed change with the DBA and plan another approach.

Let's listen in on the team's discussion.

**Tim:** I can create a PowerShell script that reads the database change file. The script provides the file's contents so that the DBA can review and approve the changes.

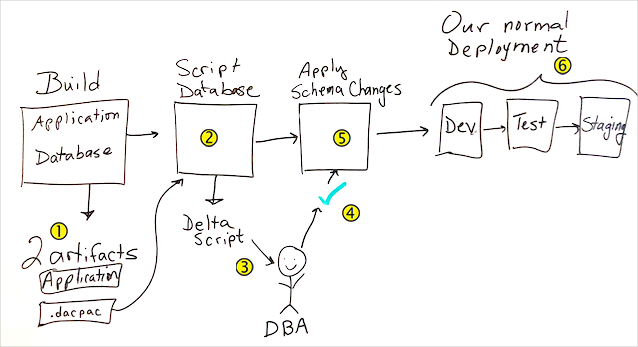
**Andy:** After the change is approved, we use SqlAzureDacpacDeployment@1 again to apply the changes.

**Mara:** That's a good place to start.

## The plan

**Andy:** So if we all agree, here's the plan.

Andy moves to the whiteboard and sketches out the plan.



**Andy:** Here's what we need to build. First we add a job to the Build stage to build the database project. Callout 1 on the whiteboard drawing. This job produces a .dacpac file that we treat as a build artifact.

We then add a stage that Callout 2 on the whiteboard drawing. scripts the database changes. The Callout 3 on the whiteboard drawing. DBA uses the script output to verify the changes before they're applied.

We add an Callout 4 on the whiteboard drawing. approval to another stage. The approval Callout 5 on the whiteboard drawing. applies the database changes. Then we Callout 6 on the whiteboard drawing. deploy to Dev, Test, and Staging just like we did before.

**Tim:** I'll get started on the PowerShell script.

**Mara:** I'll make the database project and update the website to use the database.

**Andy:** I'll get with our DBA to set up the database. But first, more coffee.